Declarative, Secure, Convergent Edge Computation

Christopher Meiklejohn
Machine Zone, Inc.
cmeiklejohn@machinezone.com

All that is new is, by that fact, automatically traditional.

T. S. Eliot

Abstract

We present an alternative model for building distributed applications that operate over replicated, shared state. This design embraces the "eventually consistent" nature of the world, focused on building computations that increase in accuracy as information is disseminated between members of a system.

1. Introduction

Many of today's mobile applications and sensor networks are still being built using a traditional client-server model, which places the data center at the center of computation. In this model, clients generate data at the edge, usually a time series of immutable events, that is then transmitted to a data center for processing by an application service provider. In the event that clients are disconnected and unable to transmit this data, two approaches have been used in practice: generated data can be buffered and transmitted once connectivity is restored or the client can prohibit the generation of events at the device by restricting application use.

This centralized processing model is highly desirable to application developers because it presents a familiar programming paradigm with an efficient system for executing computations. In concrete terms, it allows application developers to compute with data that is stored in a central location, reducing the amount of complexity inherent to computations that occur across a dynamic number of occasionally disconnected clients with varying network latencies.

While widely deployed in practice, this design is inherently limited as a result of both limited storage and power capacity on edge devices. In an ideal design, clients would be able to operate over replicated, shared state and perform some local computation: this design results in less overhead in state transmission and allows devices to make local decisions when they were disconnected from the network.

To provide a concrete example, consider the case of edge devices that monitor the temperature of a refrigerator in a hospital. While it is possible to buffer events, such as the refrigerator becoming too warm, without making local decisions at the device while the device is disconnected, the eventual delivery of buffered messages may be too late.

2. "Building On Quicksand"

When moving to a model of computation with replicated, shared state, practitioners have typically treated a centralized server, or database, as the "source of truth" for the data stored in it. In this model, clients ask for a copy of some state, perform local mutations on that state, and then attempt to write this updated state back to the server.

When there are multiple clients in the system, read and write operations are interleaved amongst the clients. Resulting from this interleaving, multiple clients can attempt to concurrently modify the same state. In a centralized approach with a single server, this is traditionally handled by a transaction protocol that only allows one of the conflicting operations to succeed.

However, when we move to a model where there are multiple replicas of some shared state, typically done to reduce latency and increase fault-tolerance, it becomes increasingly difficult to enforce a total order without reducing availability of the system. [3] Put in other words, to enforce a total order over concurrent modifications to replicated, shared state, one must synchronize with a majority of the replicas for the operation to complete; in the event that some nodes are not reachable due to network connectivity problems, the system can no longer make progress. This issue is only exacerbated as mobile and sensor network applications are deployed.

As an alternative, several "eventually consistent" database systems have been developed. These systems typically offer weaker consistency guarantees and strive for high-availability and fault-tolerance. Arguably the most famous of these systems, Amazon's Dynamo, only guaranteed that all updates would eventually be delivered to all replicas; without any ordering guarantees for events in the system, some concurrent modifications performed at different replicas would ultimately conflict and need to be resolved by the application developer. [2]

3. "Single System Illusion"

We posit that the "eventually consistent" view of the world is more compatible with these new types of applications, and more correctly models interactions between entities in the physical world.

For example, we can imagine a design where clients own the canonical copy of their data. This data is locally mutated by clients and shared to other clients in the system: when other clients in this system mutate this state, this is represented as new data that is causally related to the original data. This data between clients is disseminated in a peer-to-peer manner.

Computations, or instructions for deriving some value from this data, can also be disseminated between members of the system. More formally, computations are first class. Computations reflect the completeness of their input data: as more up-to-date information is provided, these computations should be able to be incrementally updated to improve their accuracy. Computations reflect derived knowledge based on a partial view of the system. It is important to note that this derived information knowledge can be disseminated without necessarily also supplying the source information: for instance, you might know that the Earth travels around the Sun without knowing *why*.

Computations additionally compute causality, or a notion of provenance. The result of a computation contains a record of the

input values that contributed to output. This information allows the results of computations to be partially ordered, comparable, and mergeable.

4. "On The Road To Find Out"

We previously proposed a initial solution to the problem of large-scale distributed programming without coordination, named Lasp. [8] Lasp uses declarative, functional programming techniques to deterministically compose conflict-free replicated data types, which model sequential data structures that when distributed, guarantee convergence under concurrent mutations, out-of-order message delivery and limited connectivity. This gives applications developed in Lasp a strong convergence property: given replicated state that is concurrently edited and eventually communicated to every node in a distributed system, regardless of ordering, distributed applications will converge to the correct result.

We subsequently extended Lasp by introducing a distributed, epidemic-based distribution model. [9] This distribution model uses an efficient epidemic broadcast protocol [5] to support a large number of nodes, with out-of-order message delivery and pairwise repair between nodes. Lasp exploits this distribution model for all application state, given that its use of CRDTs guarantees convergence under message replay and reordering, fundamental properties of unreliable, asynchronous networks.

Unfortunately, Lasp does not solve all of the problems. While Lasp provides the basic building blocks for building distributed computations that operate over each client's individual state, there are still many fundamental problems to solve.

One problem is causality. Causality is important for the incremental maintenance and mergeability of computation results. For example, if we have a replicated, shared set and we compute some function over that set, how can we efficiently represent the input to the computation in the output, as this allows us to incrementally update the output as the input changes?

Another problem is security. If we move all computation to the edge, and treat computations as first class, how can we write distributed computations where each member of the system can incrementally contribute to a final result in a way where the individual users values are not exposed? Returning to the causality example in the previous paragraph, how can we securely compare causality information to determine if values are stale?

Finally, the problem of expressiveness. What types of problems are expressible within a programming abstraction that assumes very weak ordering of events, the encoding of causality within the objects of the system, and operates using abstractions that rely on the idempotent and commutativity of mutations? Is programming this way too limiting?

5. Related Work

Most applications today that perform data processing from devices located at the edge take advantage of a MapReduce-style programming model over immutable data [1] and subsequent optimizations for efficient, fault-tolerant processing over streams. [10, 11] These solutions are appealing to the application developer: they present systems for performing efficient computation in a now-familiar programming paradigm.

Alternative techniques have been presented by academia that focus on moving computation to the edge to alleviate the need for transmission of the entire data set. Directed [4] and digest [12] diffusion presented efficient, fault-tolerant approaches for dissemination of computations and their results. However these systems do not expose a general programming model for arbitrary computations and required knowledge of the distribution protocol when implementing new computations.

Declarative approaches inspired by modern query languages, such as Tiny AGgregation, [7] have been proposed for data collection and aggregation across sensor networks. However, these approaches have presented abstractions that are not for general programming as they are specific to the details of aggregation in sensor networks.

Finally, declarative approaches have also been proposed for [6] computation in large-scale peer-to-peer systems where clients own their own data. However, in providing Turing completeness, the presented abstractions result in programmer having to work with low level language details.

6. Conclusion

As we continue to move towards application designs that operate over a large amount of client generated data in a privacy-conscious world, is it possible to shift our models of computation to the client in a secure manner? Is it possible to perform distributed computation across replicated, shared state that is tolerant to the problems of unreliable, asynchronous networks?

References

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[3] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[4] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67. ACM, 2000.

[5] J. Leitao, J. Pereira, and L. Rodrigues. Epidemic broadcast trees. In *26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 301–310. IEEE, 2007.

[6] D. H. Lorenz and B. Rosenan. Separation of powers in the cloud: where applications and users become peers. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, pages 76–89. ACM, 2015.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[8] C. Meiklejohn and P. Van Roy. Lasp: A Language for Distributed, Coordination-Free Programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*, pages 184–195. ACM, 2015.

[9] C. Meiklejohn and P. Van Roy. Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation. In *Workshop on Planetary-Scale Distributed Systems collocated with SRDS 2015*. IEEE, 2015.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[11] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pages 10–10. USENIX Association, 2012.

[12] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 139–148. IEEE, 2003.